КОМБІНОВАНИЙ КОМПЕТЕНТНИЙ ПАРАЛЕЛЬНИЙ ГЕНЕТИЧНИЙ АЛГОРИТМ ТА ЙОГО ЗАСТОСУВАННЯ ДЛЯ ЗАДАЧІ ПОБУДОВИ РОЗКЛАДІВ

Медвідь С.О.

Національний Університет «Києво-Могилянська Академія», м. Київ, вул Сковороди, 4,
mso@ukma.kiev.ua

Розглянуто компетентні генетичні алгоритми, що навчаються зв'язності, а також способи паралельної реалізації генетичних алгоритмів. Запропоновано власний алгоритм, що об'єднує методики побудови генетичних алгоритмів, та репрезентовано спосіб розв'язання NP-повної задачі складання розкладу занять в університеті.

Компетентні генетичні алгоритми

В останнє десятиріччя сталися великі зрушення у розвитку *компетентних генетичних алгоритмів*, здатних швидко, надійно та акуратно розв'язувати складні задачі [1-22]. Щодо обчислень існування компетентних генетичних алгоритмів (ГА) передбачає, що багато складних задач можна розв'язати за досить швидкий час. Більш того, з подібними алгоритмами немає потреби шукати гарне кодування або гарний генетичний оператор. Оскільки такий ГА може адаптуватися під конкретну задачу, у користувача відпадає потреба у адаптації задачі, кодування чи операторів під конкретну реалізацію ГА.

Розглянемо підклас компетентних ГА – селекторекомбінативні ГА. Проблему створення таких ГА [4, 18, 19] було декомпозовано з використанням визначення будівельного блоку, вперше запропонованого Дж. Холанд [5]. Розглянемо декомпозицію побудови такого алгоритму.

Будівельні блоки. Селекторекомбінативні ГА працюють завдяки механізму декомпозиції та повторного збору. Холанд запропонував звати будівельними блоками (ББ) гарно адаптовані набори ознак, що були компонентами ефективних розв'язків. Головна визначна особливість ГА полягає в тому, що вони (a) непомітно виділяють ББ (або підчастини хороших розв'язків) і (b) рекомбінують різні підчастини, формуючи дуже продуктивні розв'язки.

ББ-складні задачі. Складні задачі – це такі задачі, що мають ББ, які важко отримати. Це може відбуватися через те, що вони приховані або складні, їх просто важко знайти, чи тому, що важко відділити один ББ від іншого. Ще одна причина може полягати в тому, що ББ низького порядку можуть бути тупиковими або інакше *оманливими* [2]. З цього терміну виникла *задача-омана*.

Запас будівельних блоків та прийняття рішень. Однією з ролей популяції є забезпечення адекватного запасу ББ для подальшої роботи ГА [5, 20, 21]. Популяції, що були генеровані довільним чином, при збільшенні їхнього розміру з більшою ймовірністю міститимуть велику кількість ускладнених ББ. Більш того, процес прийняття рішень серед різних, конкуруючих сутностей (тобто ББ) статистичний за своєю природою, тому при збільшенні розміру популяції збільшується імовірність, що ГА прийматиме якісніші рішення в процесі еволюційного пошуку.

Ідентифікація та обмін будівельних блоків. Мабуть, найважливішим результатом дослідження компетентних ГА є той, що ідентифікація та подальший обмін ББ у популяції є необхідною умовою успіху еволюційного процесу у ГА. Класичні ГА зазвичай не могли надійно проводити такий обмін. При переході від класичного до компетентного ГА зміна полягає в умінні ідентифікувати і забезпечувати ефективний обмін ББ. Саме цілеспрямовані зміни структури ГА призвели до розробки компетентних ГА.

Цікаво, що механіка різних компетентних ГА може суттєво різнитися, але принципи, покладені в їхню основу, залишаються незмінними. Створення компетентних ГА почалося з невпорядкованого ГА (messy GA) [17], який у 1993 році було покращено до швидкого невпорядкованого ГА (fast messy GA). З того часу створено інші компетентні алгоритми, які використовують різні механізми.

– *Компетентні ГА, що використовують технологію збурення*, включають швидкий невпорядкований ГА (fmGA) [6], невпорядкований ГА генних виразів GEMGA (gene expression messy GA) [7] та алгоритм визначення зв'язності за допомогою ГА, що визначає монотонність (linkage identification by monotonicity detection GA) [22].

– *Технології адаптації зв'язності* – LLGA-алгоритм, що навчається зв'язності (Linkage Learning Genetic Algorithm) [8, 9].

– *Технології побудови імовірнісної моделі* – розширений компактний ECGA-алгоритм (extended compact GA), IDEA-алгоритм оцінки ітераційного розподілу (iterated distribution estimation algorithm), BOA-алгоритм баєсівської оптимізації (Bayesian optimization algorithm) та hBOA-алгоритм ієрархічної баєсівської оптимізації [8].

Алгоритми, що навчаються зв'язності

Розглянемо механізми LLGA-алгоритму [8]. На задачах, що мають експоненційні ББ, LLGA автономно пересортовує гени в хромосомі так, що пов'язані один з одним гени опиняються поруч – стають компактними. Після того, як усі зв'язки у хромосомі компактні, у LLGA немає проблем з розмноженням ББ. На подібних задачах LLGA ефективно вивчає зв'язність кожного ББ, одного за іншим. Спочатку вивчається зв'язність найбільш важливого ББ, і щойно він сформований, алгоритм приступає до наступного за значенням ББ тощо. При цьому не втрачається різноманітність геному в популяції завдяки використанню механізму ймовірнісних виразів (probabilistic expression mechanism) та оператору кросоверу.

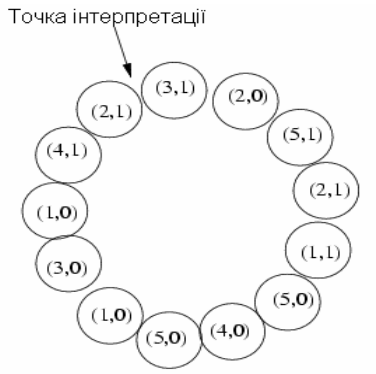

Рис. 1 – Приклад хромосоми, що використовується у LLGA

Подання даних у LLGA про кожен ген складає пару (локус, алель). Кожен ген завжди має в хромосомі цю пару значення. Одне з них виражається та передається функції придатності за ймовірнісною схемою. Утримання в хромосомі комплементарної копії кожного гену запобігає втраті генетичного різноманіття. На додаток, хромосома містить некодуючі гени (інтрони), що не впливають на функцію придатності. Їхнє призначення – забезпечити розмноження ББ та приймати участь у формуванні зв'язності. Вивчення ефекту використання некодуючого матеріалу у хромосомах [10, 11] вмотивовано тим, що некодуюча ДНК існує у живих організмів. Зокрема приблизно 97% ДНК людини складається з інтронів.

Хромосому, що використовує LLGA, зручно подавати у вигляді кільця. Розглянемо приклад кодування трибітної задачі. На рис. 1 гени 1, 2 та 3 – кодуючі, гени 4, 5 – некодуючі.

У кожної хромосоми є точка інтерпретації. Гени читаються, починаючи з цієї точки, за годинниковою стрілкою. При цьому записується значення першого зустрітого гена, усі інші гени з таким же локусом ігноруються. В прикладі на рис. 1 отримаємо (3,1)(2,0)(1,1). Ці три гени буде відправлено функції придатності. Гени 4 та 5 не будуть відправлені цій функції, бо вони є некодуючими. Зауважимо, що для кожного гена в хромосомі присутня щонайменше одна його копія.

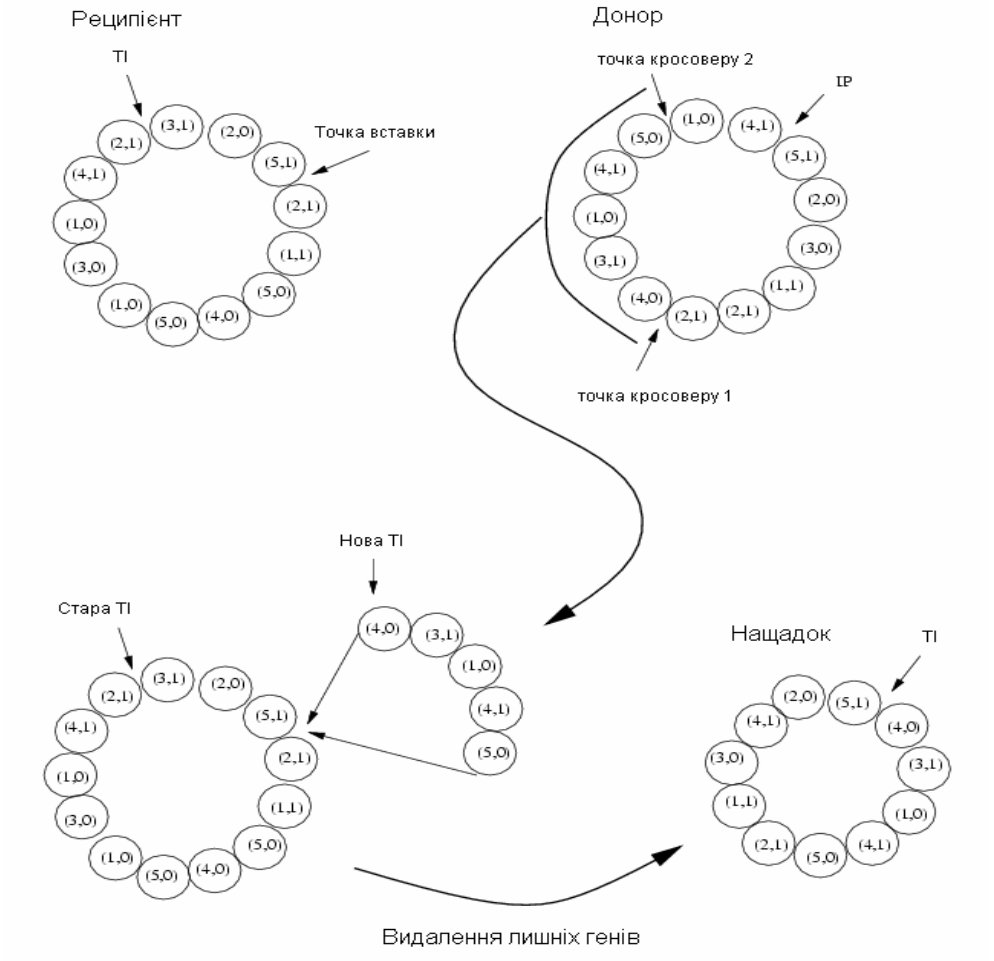

Рис. 2 – Кросовер у LLGA

Таке подання зветься імовірнісним виразом, оскільки одна хромосома може задавати набір різних розв'язків залежно від положення точки інтерпретації. Щойно наведений приклад – це розширений імовірнісний вираз EPE-2 (Extended Probabilistic Expression). Двійка в абревіатурі означає, що кожен ген має одну копію, що експресується, та максимум дві прихованих копії [8].

Кросовер як оператор у LLGA працює наступним чином. По-перше, з популяції обираються дві довільних хромосоми: донор і реципієнт. З донорської хромосоми обирається довільний сегмент та вставляється в довільно обрану точку реципієнта. Хромосома-реципієнт перевіряється за годинниковою стрілкою і з неї видаляються усі зайві гени. Після цього реципієнт стає знову легальною EPE-2 хромосомою (рис. 2). У разі правильного обрання тиску відбору, такий оператор кросоверу результує у меншій відстані між ББ. Причини цього докладніше описані у [8]. Коли зв'язність хромосоми стає високою, ББ має дуже невеликі шанси бути знищеним, а тому легко переходить до інших розв'язків у популяції.

Паралельні генетичні алгоритми

Головна ідея більшості паралельних алгоритмів – розділити завдання на частини та розв'язувати їх одночасно, використовуючи багато процесорів. Цей метод „розділяй та володарюй" можна використати багатьма способами, а тому це призводить до багатьох способів паралелізації ГА. Деякі з них змінюють поведінку ГА. Деякі методи доцільніші для масово-паралельних комп'ютерних систем, а інші – у мережі з меншою кількістю більш потужних вузлів. Класифікація паралельних ГА, що наведена у даному розділі, схожа на інші класифікації [12, 13, 14].

Першим способом є глобальна паралелізація ГА . В цьому типі паралельних ГА протягом їхньої роботи існує тільки одна популяція, як і в непаралельній реалізації, але генетичні оператори та функції придатності розраховуються паралельно. Загалом алгоритм концептуально не відрізняється від стандартного ГА. Порівняно з однопроцесорною версією швидкість може досить суттєво збільшитися за умови, що час комунікації не перевищує час обчислення [15].

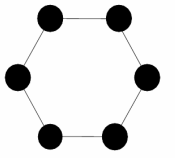

Рис. 4. Схема крупнозернистого паралельного ГА

Цікавіша ідея лежить в основі крупнозернистого паралельного ГА (рис. 3), коли популяція розділяється на багато підпопуляцій, які еволюціонують незалежно одна від одної більшість часу, періодично обмінюючись індивідами. Такий обмін кращими індивідами зветься міграцією і контролюється кількома параметрами. Крупнозернистий паралельний ГА вносить суттєві зміни в роботу ГА та має відмінну від звичайного ГА поведінку. Інколи крупнозернистий паралельний ГА називають „розподіленим" ГА, тому що його часто реалізують на MIMD-комп'ютерах з розподіленою пам'яттю. Крупнозернисті ГА також відомі як „острівкові" паралельні ГА, оскільки у популяційній генетиці описано модель структури популяції, що схожа на наведену на малюнку.

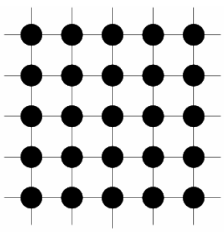

Рис. 3. Дрібнозернистий паралельний ГА

Третій підхід паралелізації ГА використовує дрібнозернистий паралелізм, коли популяція розділяється на велику кількість дуже малих популяцій. Ідеальним випадком є така конфігурація, в якій на кожен процесор припадає лише одна хромосома (рис.4). Така модель розпалалелювання призначена для масивно-паралельних комп'ютерів, але вона також корисна у будь-якому багатопроцесорному середовищі.

Зазначимо, якщо застосування методу глобальної паралелізації не впливає на поведінку алгоритму, останні два методи вносять фундаментальні зміни у спосіб роботи ГА. Зокрема при використанні методів, що розділяють популяцію на частини, парування кожної хромосоми можливе тільки з хромосомами з тієї ж підпопуляції. Останній метод паралелізації ГА використовує певну комбінацію перших трьох, його ще називають гібридним паралельним алгоритмом.

Комбінований компетентний паралельний генетичний алгоритм

Розглянемо реалізацію комбінованого компетентного паралельного ГА (ККПГА), в якому поєднано методики автоматичного налаштовування параметрів та алгоритм, що навчається зв'язності. Накладення цих двох алгоритмів реалізовано у спеціально розробленому фреймворку. Основна ідея полягає у тому, що фреймворк визначає основні спільні інтерфейси, які використовуються всіма ГА. До них належать інтерфейси до:

- операторів зміни хромосоми, репродукції і відбору;
- класів, що визначають стратегію еволюції;
- до структур даних – алелей, генів та хромосом.

Така структуризація уможливлює реалізацію ГА, код яких оперує цими інтерфейсами, а не конкретними класами. При розв'язанні реальної задачі на вхід програми подається конкретний клас, здатний породжувати інші конкретні класи. У такий спосіб клас, що визначає реальну задачу, можна написати будь-коли та інкорпорувати у реалізацію фреймворку. При цьому встановлення параметрів може відбуватися динамічно, це справджується як для класичних параметрів (імовірності спрацьовування генетичних операторів), так і для параметрів, що обирають стратегію алгоритму.

Автоматичне налаштовування параметрів. Існує два підходи до автоматичного налаштовування параметрів: централізований і децентралізований. У нашій реалізації алгоритму централізоване керування параметрами відбувається на рівні класів, що реалізують інтерфейси стратегії еволюції. Використовуючи допоміжні класи, що ведуть історію розвитку популяції, ККПГА визначає ефективність кожного з операторів і на підставі такої статистики додає до них або віднімає значення переваги використання. Оператор з низькою перевагою застосовується в процесі еволюції рідше, що призводить до балансування значень параметрів протягом еволюційного процесу.

Алгоритм, що навчається зв'язності. Для реалізації цього алгоритму насамперед до фреймворку слід додати клас, який керує стратегією розвитку популяції і розширює стандартний інтерфейс `IEvolution`. Оскільки алгоритм, що навчається зв'язності, оперує спеціальними типами даних, не схожими на ті, що використовуються у класичних ГА, то необхідно також розширити інтефейси основних структур даних – гена та хромосоми. Використання патерну „фабричний метод" уможливлює створення цих структур даних понад якимись іншими структурами, що успадковують стандартні інтерфейси.

Скажімо, клас `LLGAChromosome` реалізований так, що приймає як параметр тип `IChromosome` (тобто будь-яку іншу хромосому, яка може бути на даний момент ще й не реалізована) і сам успадковує цей інтерфейс. Це дозволяє побудувати його поверх іншого типу хромосоми (наприклад, класичного представника хромосом `BitStringChromosome`) і одночасно мати змогу використовувати у будь-якій реалізації алгоритму, що передбачає роботу зі стандартними хромосомами.

Перехід до паралельної реалізації. Розглянемо спосіб переведення отриманого алгоритму до паралельної реалізації. Паралельні реалізації якогось з ГА (глобально-паралельний, крупнозернистий і дрібнозернистий) передбачають програмування низки допоміжних класів, призначення яких – займатися передачею, комутацією та іншими видами керування віддалених викликів.

Глобально-паралельний ГА. За наявності згаданої інфраструктури у фреймворку, найпростішим для реалізації є глобально-паралельний ГА. Призначенням додаткового коду у такому алгоритмі є рівномірний перерозподіл навантаження, створюваного основними розрахунками, що ведуться ГА. Як видно з опису класичного ГА, найбільше обчислювальне навантаження припадає на розрахунки, пов'язані з роботою генетичних операторів.

Найбільш природнім для реалізації у межах визначеного фреймворку є спадкування основних інтерфейсів, спільних для генетичних операторів та хромосом, і прийняття реальних операторів та хромосом як параметрів при створенні класу. Так, клас `GPChromosome` (що означає Global Parallel Chromosome), приймає у своєму конструкторі тип `IChromosome` і одночасно успадковує його ж.

Розпаралелювання навантаження виконує клас `GPManager`, з яким взаємодіє кожен екземпляр `GPChromosome`. Маючи інформацію про вузли мережі, що співпрацюють із `GPManager`, останній відправляє їм екземпляри хромосом, надані `GPChromosome`. При зверненні варіанту реалізації ГА, що працює у цей момент, до класу `GPChromosome`, останній, використовуючи `GPManager`, робить віддалений виклик через мережу до комп'ютера, якій містить реальний екземпляр хромосоми, потрібної ГА. Перевага такої реалізації розпаралелювання – незалежність від уже створених ГА, що працюють у фреймворку і не здогадуватимуться про те, що насправді вони працюють у розподіленому середовищі.

Інші паралельні реалізації. Реалізація інших паралельних алгоритмів в рамках розробленого фреймворку принципово не відрізняється від глобально-паралельного ГА. Основна ідея будь-якого розпаралелювання – це написання коду, що реалізує спільні інтерфейси, перехоплюючи потоки даних, які циркулюють у ГА. Перехопивши, кожна конкретна реалізація паралельного ГА вчиняє згідно із стратегією, закладеною у її алгоритм. Скажімо, як для дрібнозернистого паралельного ГА, так і для крупнозернистого зручно успадковувати інтерфейс `IPopulation`, щоб мати повний контроль над структурою популяції. Зокрема можна вибирати певні хромосоми з популяції та відправляти їх на інший сервер, під'єднаний до обчислювального середовища.

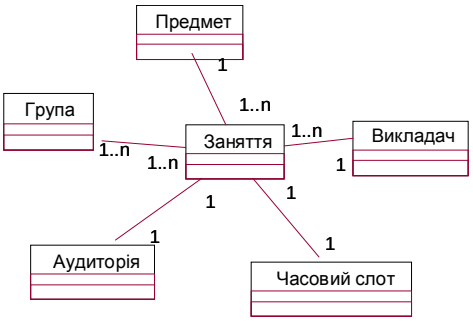

Рис. 5 – Основні сутності, з яких складається розклад

Реальна задача складання розкладу занять в університеті

Оптимізаційні задачі, що виникають на практиці, часто відрізняються надвеликою складністю від тих, на основі яких розвивається теорія ГА. Розглянемо одну з таких задач складання розкладу в університеті, яка є NP-повною задачею [22] надвеликої розмірності.

Передусім визначимося із формальною постановкою задачі, тоді зазначимо особливості реалізації розширень до алгоритму, що був запропонований у цій роботі, та встановимо методи дослідження його ефективності (рис. 5).

Основні частини розкладу. Складання розкладу передбачає призначення часу проведення пар для заданих груп студентів у заданих аудиторіях. Визначимо сутності, якими оперуватиме алгоритм.

– Група має усталений список студентів та є неподільною одиницею розкладу.

– Предмет – це навчальний курс, що його слухають групи.

– Викладач – людина, що веде заняття.

– Часовий слот – абсолютний номер пари на тижні. Якщо розглядати 6 часових слотів на день, то для робочого тижня отримаємо 30 часових слотів (див. табл.1).

– Аудиторія – місце проведення заняття.

– Заняття (можна назвати парою) – сутність, що пов'язує усі інші у тижневому розкладі. Зокрема:

– предмет може мати одне або кілька занять на тиждень. При цьому не має значення тип заняття (лекція, практична);

– один викладач відповідає за проведення одного або кількох занять за тиждень;

– заняття може займати лише один часовий слот протягом тижня;

– заняття може проходити лише в одній аудиторії;

– група студентів може слухати одне або кілька занять протягом тижня, а також одне заняття може слухати одна або кілька груп (останнє справджується у потокових лекціях).

Таблиця 1 – Абсолютні номери часових слотів

Номер пари	Пн	Вт	Ср	Чт	Пт
I	1	7	13	19	25
II	2	8	14	20	26
III	3	9	15	21	27
IV	4	10	16	22	28
V	5	11	17	23	29
VI	6	12	18	24	30

Початкові дані. *На вході алгоритму складання розкладу задаються наступні зв'язки:*

– заняття – викладач;

– заняття – групи, що його слухають;

– заняття – предмет.

Отже, алгоритм має маніпулювати зв'язками „заняття – аудиторія" і „заняття – часовий слот". Це зручно подати у вигляді табл.. 2.

Таблиця 2 – Подання даних, якими маніпулює ГА

Часовий слот	Ауд. 1	Ауд. 2	Ауд. 3	Ауд. 4	...
1	Заняття 4	Заняття 14	Заняття 7	Заняття 1	...
2	Заняття 1	Заняття 6	Заняття 3	(пусто)	...
3	Заняття 2	(пусто)	Заняття 12	Заняття 9	...
4	(пусто)	(пусто)	(пусто)	Заняття 10	...
5	Заняття 11	Заняття 15	Заняття 13	Заняття 8	...
6	(пусто)	Заняття 5	(пусто)	Заняття 1	...
...	...	...	...	...	...

У цій таблиці рядки вказують часові слоти протягом тижня, а стовпчики – аудиторії. На перетинах записується абсолютний номер заняття. Так, для навчального закладу, що має 45 аудиторій, 40 груп, кожна з яких слухає по 20 занять на тиждень, розмірність таблиці буде 45x30 = 1350 клітин, з них 40x20 = 800 клітин буде відведено під заняття, а 1350-800 = 550 залишаться вільними. Тоді задачею ГА буде розташування 800 елементів на 1350-ти місцях, що складає (1350!)/(550!) ~ $3x10^{2371}$ варіантів – надзвичайно велике число, а цифри наведені щодо невеликої школи!

Обмеження, що накладаються на розклад. Будемо розглядати наступні жорсткі обмеження, порушуючи які розклад стає неприйнятним розв'язком задачі:

– одна група не може одночасно знаходитися в кількох аудиторіях;

– одне заняття не може проводитися одночасно в кількох аудиторіях;

– один викладач не може одночасно проводити кілька занять.

Кодування хромосоми. Розглянемо особливості кодування задачі складання розкладу занять у хромосому. Як відомо, задавши таке кодування, а також фунцію придатності для хромосоми, в подальшому можна абстрагуватися від задачі, оскільки ГА працюватиме з типовою задачею оптимізації чорної скриньки. Для кодування табл. 2 у хромосому випишемо заняття в рядок у порядку зростання номерів. Це і будуть наші будівельні блоки у хромосомі. Їхня довжина залежатиме від розмірності задачі, оскільки значення цього будівельного блоку відповідатиме порядковому номеру клітини, в яку записано заняття. Зокрема для нашого прикладу кількість клітин складає 1350, а для запису цього значення знадобиться 11 бітів. Тобто ББ хромосоми при розв'язанні цієї задачі матиме розмір 11 генів.

Оскільки для більшості реальних задач кількість клітин у табл. 1 не буде дорівнювати степені двійки, і якщо ББ хромосоми подано двійковим значенням номеру клітини, то не всі його значення кодуватимуть реальний номер клітини у таблиці (зокрема, блок, що складається тільки з одиниць, майже ніколи не буде приймати участь в еволюції). Тому введемо вироджений код, який в деяких кодуватиме одну клітину таблиці максимум двома значеннями ББ.

Підрахунок функції придатності хромосоми. Для підрахування функції придатності хромосоми, що кодує розклад занять, будемо поступово проходити по хромосомі, беручі гени групами розміром з ББ. Кожен ББ кодуватиме заняття з певним порядковим номером, до якого однозначно прив'язаний список груп, та викладач. За порядковим номером ББ однозначно встановлюються час і місце заняття.

Отриманої інформації достатньо, щоб відновити з хромосоми розклад, що був у ній закодований. Тому цю інформацію можна використовувати у розрахунках функції придатності, використовуючи поняття жорстких та нежорстких обмежень, як про це було докладно описано на початку цієї роботи.

Висновки

Головні напрямки сучасних досліджень з оптимізації ГА – побудова компетентних ГА та їхня паралелізація Саме ці напрямки стали визначальними у побудові власної реалізації комбінованого алгоритму, найвдалішими рисами якого є автоматична адаптація параметрів та механізм навчання зв'язності.

Для паралелізації ГА як перспективного шляху збільшення продуктивності алгоритму, особливо з розвитком мережних комунікацій, розглянуто чотири основні способи: глобально-паралельний, крупнозернистий, дрібнозернистий та гібридний. Перший з них було реалізовано у власному алгоритмі.

Окреме дослідження полягало у застосуванні отриманого ГА до реальної задачі складання розкладу в навчальному закладі. Від попередньої реалізації алгоритму [1] нова реалізація суттєво відрізняється насамперед новим кодуванням хромосом і способом підрахунку функції придатності хромосом. Розроблено фреймворк для тестування та роботи з ГА.

Джерела

1. Глибовец Н.Н., Медведь С.А. Генетические алгоритмы и их использование для решения задачи составления расписаний //Кибернетика и системный анализ. 2003. - № 1. - С. 95-108.
2. Goldberg D.E. Simple genetic algorithms and the minimal, deceptive problem //Genetic Algorithms and Simulated Annealing. - Los Altos, CA: Morgan Kaufmann, 1987. - P. 74-88.
3. Goldberg D. E., Deb K., Clark J. H. Genetic algorithms, noise, and the sizing of populations //Complex Systems. - 1992. - No 6. - P. 333–362.
4. Holland J. H. Adaptation in natural and artificial systems. - Ann Arbor, MI: University of Michigan Press, 1975. – 97 p.
5. Rapid, accurate optimization of difficult problems using fast messy genetic algorithms / D.E.Goldberg , K.Deb, H.Kargupta, G.Harik //Proceedings of the Fifth International Conference on Genetic Algorithms. - San Mateo, CA: Morgan Kaufmann, 1993. - P. 56–64.
6. Kargupta H. The gene expression messy genetic algorithm //Proceedings of 1996 IEEE International Conference on Evolutionary Computation. - New York: IEEE Press, 1996. - P. 814–819.
7. Harik G. R. Learning gene linkage to efficiently solve problems of bounded difficulty using genetic algorithms. Doctoral dissertation. - Ann Arbor: University of Michigan, 1997. (IlliGAL Report No. 97005. – 86 p.).
8. Harik G. R.Linkage Learning via Probabilistic Modeling in the ECGA. - Urbana: University of Illinois at Urbana-Champaign, 1999. (IlliGAL Report No. 99010. – 112 p.).
9. Compressed Introns in a Linkage Learning Genetic Algorithm. Genetic Programming / F.G.Lobo, K.Deb, D.E.Goldberg, G.R.Harik, L. Wang //Proceedings of the Third Annual Conference, 1998. (IlliGAL report No. 97010)
10. Levenick J.R. Inserting introns improves genetic algorithm success rate: Taking a cue from biology //Proceedings of the Fourth International Conference on Genetic Algorithms. - San Mateo, CA: Morgan Kaufmann, 1991. - P. 123–127.
11. Levenick J. R. Metabits: Generic endogenous crossover control //Proceedings of the Sixth International Conference on Genetic Algorithms. - San Mateo, CA: Morgan Kaufmann, 1995. - P. 88–95.
12. Adamidis P. Review of parallel genetic algorithms bibliography (Tech. Rep. Version 1). - Thessaloniki, Greece: Aristotle University of Thessaloniki, 1994.
13. Gordon V.S., Whitley D. Serial and parallel genetic algorithms as function optimizers //Proceedings of the Fifth International Conference on Genetic Algorithms. - San Mateo, CA: Morgan Kaufmann, 1993. - P. 177-183.
14. Lin S.-C., Punch W., Goodman E. Coarse-grain parallel genetic algorithms: Categorization and new approach //In Sixth IEEE Symposium on Parallel and Distributed Processing. - Los Alamitos, CA: IEEE Computer Society Press, 1994.
15. Cant'u-Paz E. Designing efficient master-slave parallel genetic algorithms. - Urbana, IL: University of Illinois at Urbana-Champaign, 1997. (IlliGAL Report No. 97004).
16. Goldberg D.E., Korb B., Deb K. Messy genetic algorithms: Motivation, analysis, and first results //Complex Systems.- 1989.- No 3(5). - P. 493–530.
17. Goldberg D. E. The race, the hurdle, and the sweet spot: Lessons from genetic algorithms for the automation of design innovation and creativity //Evolutionary Design by Computers - San Mateo, CA: Morgan Kaufmann, 1999. - P. 105–118.
18. Goldberg D. E. The Wright Brothers, genetic algorithms, and the design of complex systems //Proceedings of the Symposium on Neural-Networks: Alliances and Perspectives in Senri 1993. - Osaka, Japan: Senri International Information Institute, 1993. - P. 1-7.
19. Goldberg D. E. Sizing populations for serial and parallel genetic algorithms //Proceedings of the Third International Conference on Genetic Algorithms, 1989. (IlliGAL Report No. 88004 - P. 70–79).
20. Goldberg D. E., Sastry K., Latoza T. On the supply of building blocks //Proceedings of the Genetic and Evolutionary Computation Conference, 2001. (IlliGAL Report No. 2001015 - P. 336–342).
21. Munetomo M., Goldberg D.E. Linkage identification by non-monotonicity detection for overlapping functions //Evolutionary Computation. – 1999. - No.7 (4). - P. 377–398.
22. Evan S., Itai A., Shamir A. On the Complexity of Timetable and Multicom-modity Flow Problems //SIAM Journal of Computing. - 1976. - V. 5. - No.4. - P. 691-703